Code library

Yifei Shan

2021 年 12 月 8 日

目录

1 基本算法 **1**
- 1.1 位运算 . . . 1
- 1.2 前缀和与差分 . . . 1
  - 1.2.1 前缀和 . . . 1
  - 1.2.2 差分 . . . 1
- 1.3 大整数 . . . 1
  - 1.3.1 大数 . . . 1
  - 1.3.2 加法 . . . 3
  - 1.3.3 减法 . . . 3
  - 1.3.4 乘法 . . . 4
  - 1.3.5 除法 . . . 4
- 1.4 区间和 . . . 4
- 1.5 set 技巧 . . . 5

2 图论 **5**
- 2.1 链式前向星 . . . 5
- 2.2 树的 dfs 序 . . . 5
- 2.3 树的深度 . . . 6
- 2.4 图的连通块划分 . . . 6
- 2.5 欧拉路 . . . 6
- 2.6 强连通分量 . . . 6
- 2.7 点分治 . . . 6
- 2.8 树的重心 . . . 7
- 2.9 树的直径 . . . 7
  - 2.9.1 两次 dfs . . . 7
  - 2.9.2 树形 dp . . . 8
- 2.10 拓扑排序 . . . 8
- 2.11 dijkstra . . . 9
  - 2.11.1 朴素算法 . . . 9
  - 2.11.2 堆优化 . . . 10
- 2.12 SPFA . . . 10
  - 2.12.1 求最短路 . . . 10
  - 2.12.2 判断负环 . . . 10
  - 2.12.3 差分约束 . . . 10
- 2.13 Floyd . . . 10
  - 2.13.1 求关系闭包 (递闭包) . . . 10
  - 2.13.2 多源最短路 . . . 11
- 2.14 LCA(最近公共祖先) . . . 11
- 2.15 二分图匹配（染色法） . . . 12
- 2.16 二分图最大匹配（匈牙利算法） . . . 12
- 2.17 Tarjan 求割边 . . . 13
- 2.18 Tarjan 求割点 . . . 13
- 2.19 网络流 . . . 13
  - 2.19.1 最大流 . . . 13
  - 2.19.2 最小割 . . . 14
  - 2.19.3 有向无环图最小路径覆盖 . . . 15

3 数据结构 **16**
- 3.1 并查集 . . . 16
- 3.2 分块 . . . 18
- 3.3 树状数组 . . . 18
- 3.4 线段树 . . . 18
- 3.5 bfs . . . 20
  - 3.5.1 小猫爬山 . . . 20
  - 3.5.2 八皇后 . . . 20

4 数论 **20**
- 4.1 筛质数 . . . 20
  - 4.1.1 线性筛 . . . 20
- 4.2 质因数分解 . . . 20
- 4.3 最大质因数 . . . 21
- 4.4 正因数集合 . . . 21
- 4.5 逆元 . . . 21
  - 4.5.1 费马小定理 . . . 21
  - 4.5.2 线性逆推 . . . 21
  - 4.5.3 扩展欧几里得 . . . 21

5 计算几何 **21**
- 5.1 点初始模板 . . . 21
- 5.2 一些基本运算 . . . 21
- 5.3 点模板总结 . . . 22
- 5.4 线 . . . 23

6 字符串 **23**
- 6.1 函数 . . . 23
- 6.2 Manacher . . . 24
- 6.3 KMP . . . 24

7 背包 **24**
- 7.1 01 . . . 24
- 7.2 完全背包 . . . 24
- 7.3 多重背包 . . . 24
- 7.4 分组背包 . . . 25

1 基本算法

1.1 位运算

```
(n >> k) & 1;  //取n的第k位
n & ((1 << k) - 1)  //取前k位
n ^ (1 << k);   //k位取反
n | (1 << k);  //第k位取1
n & (~(1 << k))  //第k位取0
```

1.2 前缀和与差分

1.2.1 前缀和

```
#include <iostream>
using namespace std;

const int N = 100010;
int n, m;
int a[N], s[N];

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i ++ ) scanf("%d", &a[i]);

    for (int i = 1; i <= n; i ++ ) s[i] = s[i - 1] + a
        [i]; // 前缀和的初始化

    while (m -- )
    {
        int l, r;
        scanf("%d%d", &l, &r);
        printf("%d\n", s[r] - s[l - 1]); // 区间和的计
            算
    }

    return 0;
}
```

1.2.2 差分

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int N=1e6+10;
int p[N],tmp[N];
long long ans;
int b[N],a[N];
void insert(int l,int r,int c)
{
    b[l]+=c;
    b[r+1]-=c;
}

int main()
{
    int t,n;
    cin>>n>>t;
    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1;i<=n;i++) insert(i,i,a[i]);
    while(t--)
    {
        int x,y,c;
        cin>>x>>y>>c;
        insert(x,y,c);
    }
    for(int i=1;i<=n;i++) b[i]+=b[i-1];
    for(int i=1;i<=n;i++) cout<<b[i]<<" ";
}
```

1.3 大整数

1.3.1 大数

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct BigInteger{
    //BASE为vector数组中一位中最大存储的数字，前面都是以
        10计算的
    //WIDTH为宽度
    static const int BASE = 100000000;
    static const int WIDTH = 8;
    vector<int> s;
    //正数为1，负数为-1
    int flag = 1;

    //构造函数
    BigInteger(ll num = 0){*this = num;}
    BigInteger(string str){*this = str;}
    BigInteger(const BigInteger& t){
        this->flag = t.flag;
        this->s = t.s;
    }
    //赋值函数
    BigInteger operator = (ll num){
        s.clear();
        do {
            s.push_back(num % BASE);
            num /= BASE;
        }while(num > 0);
        return *this;
    }
    BigInteger operator = (string &str){
        s.clear();
        int x,len = (str.length()-1)/WIDTH + 1;
        for(int i=0;i<len;i++){
            int end = str.length() - i*WIDTH;
            int start = max(0,end - WIDTH);
            sscanf(str.substr(start,end-start).c_str(),
                "%d",&x);
            s.push_back(x);
        }
        return *this;
    }

    //基本比较函数 A < B
    bool cmp( vector<int> &A, vector<int> & B){
        if(A.size() != B.size())return A.size() < B.
            size();
        for(int i=A.size()-1;i>=0;i--){
            if(A[i] != B[i]){
                return A[i] < B[i];
```

```
            }
        }
        return false;
    }
    //比较函数如果小于则返回真
    bool operator < ( BigInteger & b){
        return cmp(s,b.s);
    }
    bool operator > ( BigInteger& b){
        return b < *this;
    }
    bool operator <= ( BigInteger &b){
        return !(b < *this);
    }
    bool operator >= ( BigInteger &b){
        return !(*this < b);
    }
    bool operator == ( BigInteger &b){
        return !(b < *this) && (*this < b);
    }
    //基本四则运算
    vector<int> add(vector<int> &A, vector<int> &B);
    vector<int> sub(vector<int> &A, vector<int> &B);
    vector<int> mul(vector<int> &A, int b);
    vector<int> mul(vector<int> &A, vector<int> &B);
    vector<int> div(vector<int> &A, int b);
    vector<int> div(vector<int> A, vector<int> B);

    //重载运算符
    BigInteger operator + (BigInteger &b);
    BigInteger operator - (BigInteger &b);
    BigInteger operator * (BigInteger &b);
    BigInteger operator * (int& b);
    BigInteger operator / (BigInteger & b);
    BigInteger operator / (int b);
};
//重载<<
ostream& operator << (ostream &out,const BigInteger&
    x) {
    if(x.flag == -1)out << '-';
    out << x.s.back();
    for(int i = x.s.size() - 2; i >= 0;i--){
        char buf[20];
        sprintf(buf,"%08d",x.s[i]);//08d此处的8应该与
            WIDTH一致
        for(int j=0;j<strlen(buf);j++)out<<buf[j];
    }
    return out;
}
//重载输入
istream& operator >> (istream & in,BigInteger & x){
    string s;
    if(!(in>>s))return in;
    x = s;
    return in;
}
vector<int> BigInteger::add( vector<int> &A, vector<
    int> &B){
    if(A.size() < B.size())return add(B,A);
    int t = 0;
    vector<int> C;
    for(int i=0;i<A.size();i++){
        if(i<B.size())t += B[i];
        t += A[i];
        C.push_back(t%BASE);
        t /= BASE;
    }
    if(t)C.push_back(t);
    while(C.size() > 1 && C.back() == 0)C.pop_back();
    return C;
}
vector<int> BigInteger::sub( vector<int> &A, vector<
    int> &B){
    vector<int> C;
    for(int i=0,t=0;i<A.size();i++){
        t = A[i] - t;
        if(i<B.size())t -= B[i];
        C.push_back((t+BASE)%BASE);
        if(t < 0)t = 1;
        else t = 0;
    }
    while(C.size() > 1 && C.back() == 0)C.pop_back();
    return C;
}
vector<int> BigInteger::mul(vector<int> &A,int b){
    vector<int> C;
    int t = 0;
    for(int i = 0;i < A.size() || t; i++){
        if(i < A.size()) t += A[i] * b;
        C.push_back(t%BASE);
        t /= BASE;
    }
    return C;
}
//大数乘大数乘法需要将BASE设置为10，WIDTH设置为1
vector<int> BigInteger::mul( vector<int> &A, vector<
    int> &B) {
    int la = A.size(),lb = B.size();
    vector<int> C(la+lb+10,0);
    for(int i=0;i<la;i++){
        for(int j=0;j<lb;j++){
            C[i+j] += A[i] * B[j];
        }
    }
    for(int i=0;i<C.size();i++){
        if(C[i] >= BASE){
            C[i + 1] += C[i] / BASE;
            C[i] %= BASE;
        }
    }
    while(C.size() > 1 && C.back() == 0)C.pop_back();
    return C;
}
//大数除以整数
vector<int> BigInteger::div(vector<int> & A,int b){
    vector<int> C;
    int r = 0;
    for(int i = A.size() - 1;i >= 0;i--){
        r = r * 10 + A[i];
        C.push_back(r/b);
        r %= b;
    }
    reverse(C.begin(),C.end());
    while(C.size() > 1 && C.back() == 0)C.pop_back();
    return C;
}
//大数除以大数
vector<int> BigInteger::div(vector<int> A,vector<int>
    B){
    int la = A.size(),lb = B.size();
```

```
    int dv = la - lb; // 相差位数
    vector<int> C(dv+1,0);
    //将除数扩大，使得除数和被除数位数相等
    reverse(B.begin(),B.end());
    for(int i=0;i<dv;i++)B.push_back(0);
    reverse(B.begin(),B.end());
    lb = la;
    for(int j=0;j<=dv;j++){
        while(!cmp(A,B)){
            A = sub(A,B);
            C[dv-j]++;
        }
        B.erase(B.begin());
    }
    while(C.size()>1 && C.back() == 0)C.pop_back();
    return C;
}
BigInteger BigInteger::operator + ( BigInteger & b){
    BigInteger c;
    c.s.clear();
    c.s = add(s,b.s);
    return c;
}

BigInteger BigInteger::operator - ( BigInteger & b) {
    BigInteger c;
    c.s.clear();
    if(*this < b){
        c.flag = -1;
        c.s = sub(b.s,s);
    }
    else{
        c.flag = 1;
        c.s = sub(s,b.s);
    }
    return c;
}
BigInteger BigInteger::operator *(BigInteger & b){
    BigInteger c;
    c.s = mul(s,b.s);
    return c;
}
BigInteger BigInteger::operator *(int& b){
    BigInteger c;
    c.s = mul(s,b);
    return c;
}
BigInteger BigInteger::operator /(BigInteger & b){
    BigInteger c;
    if(*this < b){
        c.s.push_back(0);
    }
    else{
        c.flag = 1;
        c.s = div(s,b.s);
    }
    return c;
}
BigInteger BigInteger::operator /(int b){
    BigInteger c;
    BigInteger t = b;
    if(*this < t){
        c.s.push_back(0);
    }
    else{
        c.flag = 1;
        c.s = div(s,b);
    }
    return c;
}
int main(){
    BigInteger A,B;
    cin>>A>>B;
    cout<<A+B<<endl;
    cout<<A-B<<endl;
    cout<<A*B<<endl;
    cout<<A/B<<endl;
    return 0;
}
```

1.3.2 加法

```

vector<int> add(vector<int> &A,vector<int> &B)
{
    vector<int> C;
    int t=0;
    for(int i=0;i<A.size();i++)
    {
        t+=A[i];
        if(i<B.size()) t+=B[i];
        C.push_back(t%10);
        t/=10;
    }
    if(t) C.push_back(1); //进的最后一位
    return C;
}
```

1.3.3 减法

```
#include <iostream>
#include <bits/stdc++.h>
#include <vector>
using namespace std;
const int N=1e5+10;

bool cmp(vector<int> &A,vector<int> &B)
{
    if(A.size()!=B.size())return A.size()>B.size();
    for(int i=A.size();i>=0;i--)
        if(A[i]!=B[i]) return A[i]>B[i];
    return true;
}

vector<int> sub(vector<int> A,vector<int> B)
{
    vector<int> C;
    int t=0;//进位
    for(int i=0;i<A.size()||i<B.size();i++)
    {
        if(i<A.size()) t=A[i]-t; //借一位
        if(i<B.size()) t-=B[i];
        C.push_back((t+10)%10);
        if(t<0) t=1;
        else t=0;
    }
```

```
    while(C.size()>1&&C.back()==0) C.pop_back();//删除前导零 C.back()最后一位的值; pop_back()删除最后一位;
    return C;
}

int main()
{
    string a,b;
    cin>>a>>b;
    vector<int> A,B;
    for(int i=a.size()-1;i>=0;i--) A.push_back(a[i]-'0');
    for(int i=b.size()-1;i>=0;i--) B.push_back(b[i]-'0');
    if(cmp(A,B))
    {
        vector<int> C=sub(A,B);
        for(int i=C.size()-1;i>=0;i--) cout<<C[i];
    }
    else
    {
        vector<int> C=sub(B,A);
        cout<<"-";
        for(int i=C.size()-1;i>=0;i--) cout<<C[i];
    }
}
```

1.3.4 乘法

```

#include <iostream>
#include <vector>

using namespace std;

vector<int> mul(vector<int> &A, int b)
{
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size() || t; i ++ )
    {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }

    return C;
}

int main()
{
    string a;
    int B;
    vector<int> A;
    cin >> a >> B;
    for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');

    auto C = mul(A, B);

    for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];
    cout << endl;

    return 0;
}
```

1.3.5 除法

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

vector<int> div(vector<int> &A, int b, int &r)
{
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i -- )
    {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

int main()
{
    string a;
    vector<int> A;

    int B;
    cin >> a >> B;
    for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');

    int r;
    auto C = div(A, B, r);

    for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];

    cout << endl << r << endl;

    return 0;
}
```

1.4 区间和

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> PII;
const int N = 3e6 + 10;
int a[N], s[N];
vector<int> alls;
vector<PII> add, query;

int find(int x){
      int l = 0, r = alls.size() - 1;
      while(l<r){
```

```
            int mid = (l + r) >> 1;
            if(alls[mid] >= x) r = mid;
            else l = mid + 1;
        }
        return r + 1;
}
int main()
{
        int n, m ;
        while(~scanf("%d%d", &n,&m)){
            for(int i = 0; i < n; i++){
                int x, c;
                scanf("%d%d", &x, &c);
                add.push_back({x, c});
                alls.push_back(x);
            }
            for(int i = 0; i < m; i++){
                int l, r;
                scanf("%d%d", &l, &r);
                query.push_back({l, r});
                alls.push_back(l);
                alls.push_back(r);
            }
            //排序，去重
            sort(alls.begin(), alls.end());
            alls.erase(unique(alls.begin(), alls.end
                ()), alls.end());
            //处理输入
            for(auto item : add){
                int x = find(item.first);
                a[x] += item.second;
            }
            //前缀和
            for(int i = 1; i <= alls.size(); i++) s[
                i] = s[i-1] + a[i];
            //处理询问
            for(auto item : query){
                int l = find(item.first), r =
                    find(item.second);
                printf("%d\n", (s[r] - s[l-1]));
            }
        }

}
```

1.5 set 技巧

更改排序方式

由于 set 默认从小到大排序，若需要递减集合，更改 cmp 函数

```
//一元组集合
struct cmp {
    bool operator() (const int &a, const int &b) const
        {
        return lena > lenb;
    }
};
set<int> s;

//集合
struct cmp {
    bool operator() (const pair<int, int> &a, const
        pair<int, int> &b) const {
        int lena = a.second - a.first + 1;
        int lenb = b.second - b.first + 1;
        if (lena == lenb) return a.first < b.first;
        return lena > lenb;
    }
};
set<pair<int,int>, cmp> s;
```

2 图论

2.1 链式前向星

```
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdio>
#include <list>
#include <set>
#include <map>
#include <queue>
#include <stack>
#include <vector>
#include <algorithm>
#include <unsorted_map>
using namespace std;
const int maxn = 1e5 + 10;
const int INF = 2147483647;
int n, t, m, k, num, root,dcc,c[maxn],uu[maxn*2],vv[
    maxn*2];
int ans, dfn[maxn],low[maxn],du[maxn];
bool bridge[maxn],cut[maxn];
int sum;
struct Edge
{
    int to;
    int next;
    int w;
};
Edge edge[maxn<<2];
int cnt = 0;
int head[maxn];
void add(int u, int v)
{
    edge[cnt].to = v;
    edge[cnt].next = head[u];
    head[u] = cnt++;
}
```

2.2 树的 dfs 序

进入递归后和即将回溯前记录一次编号，产生长度为 2N 的 **dfs 序 **

** 特点 ** ：两次出现 **x** 的位置之间为以 **x** 为根的子树的 dfs 序

```
void dfs(int u){
    a[++m] = u;
    vis[u] = 1;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        dfs(v);
```

```
    }
    a[++m] = u;
}
```

2.3 树的深度

```
void dfs(int u){
    v[u] = 1;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        d[v] = d[u] + 1;
        dfs(v);
    }
    a[++m] = u;
}
```

2.4 图的连通块划分

```
void dfs(int u){
    vis[u] = 1;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        dfs(v);
    }
}

for(int i = 1; i <= n;i++){ //加在main()中
    if(!vis[i]){
        cnt++; //没有访问过，连通块次数加一，继续dfs
        dfs(i);
    }
}
```

2.5 欧拉路

```
void enlur(){
    st[++top] = 1;
    while(top > 0){
        int x = st[top] , i = head[x];
        while(i && vis[i]) i = nxt[i];
        if(i){
            st[++top] = ver[i];
            vis[i] = vis[i ^ 1] = 1;
            head[x] = nxt[i]; // 会修改head[x] 的值
        }else{
            top --;
            ans[++t] = x;
        }
    }
}
```

2.6 强连通分量

```
const int N = 100010, M = 1000010;
int head[N], ver[M], nxt[M], dfn[N], low[N];
int st[N], ins[N], c[N];
vector<int> scc[N];
int n, m, tot, num, top, cnt;
void add(int x, int y){
    ver[++tot] = y, nxt[tot] = head[x], head[x] = cnt;
}
void tarjan(int x){
    dfn[x] = low[x] = ++num;
    st[++top] = x, ins[x] = 1;
    for(int i=head[x];i;i=nxt[i]){
        if(!dfn[ver[i]]){
            tarjan(ver[i]);
            low[x] = min(low[x], low[ver[i]]);
        }else if(ins[ver[i]]) //有作用的横叉边以及返祖边
            low[x] = min(low[x], dfn[ver[i]]);
    }
    if(dfn[x] == low[x]){//x可以作为这个连通分量的入口
        cnt ++;int y;
        do{
            y = st[top--], ins[y] = 0;
            c[y] = cnt, scc[cnt].push_back(y);
        }while(x != y);
    }
}
```

2.7 点分治

求是否存在路径权值和为 k 的路径

```
const int N = 10000 + 5;
const int M = 20010;
int head[N], ver[M], edge[M], nxt[M];
int sz[N], del[N], dis[N], st[N], q[N], cnt, top;
int query[101], res[N];
bool has[10000010];
int n, m, tot, all, core, min_size;
void add(int x, int y, int z){
    ver[++tot] = y, nxt[tot] = head[x], edge[tot] = z,
        head[x] = tot;
}
void getsize(int x, int fa){
    sz[x] = 1;
    int mx = 0;
    for(int i=head[x];i;i=nxt[i]){
        int y = ver[i];
        if(y == fa || del[y]) continue;
        getsize(y, x);
        mx = max(mx, sz[y]);
        sz[x] += sz[y];
    }
    mx = max(mx, all - sz[x]);
    if(mx < min_size) {min_size = mx, core = x;}
}
void getdis(int x, int fa){
    if(dis[x] <= 10000000)
        st[++cnt] = x, q[++top] = x;
    for(int i=head[x];i;i=nxt[i]) {
        int y = ver[i];
        if(del[y] || y == fa) continue;
        dis[y] = dis[x] + edge[i];
        getdis(y, x);
    }
}
void get(int x){
    cnt = top = 0;
    del[x] = 1;
    dis[x] = 0;
```

```
    has[dis[x]] = true;
    q[++top] = x;
    for(int i=head[x];i;i=nxt[i]){
        int y = ver[i];
        if(del[y]) continue;
        dis[y] = edge[i];
        cnt = 0;
        getdis(y, x);
        for(int j=1;j<=cnt;j++){
            for(int k=1;k<=m;k++){
                if(query[k] >= dis[st[j]]){
                    res[k] |= has[query[k] - dis[st[j
                        ]]];
                }
            }
        }
        for(int j=1;j<=cnt;j++) has[dis[st[j]]] = true
            ;
    }
    for(int i=1;i<=top;i++) has[dis[q[i]]] = false;
    for(int i=head[x];i;i=nxt[i]){
        int y = ver[i];
        if(del[y]) continue;
        min_size = inf; all = sz[y];
        getsize(y, 0);
        getsize(core, 0);
        get(core);
    }
}
int main(){
    scanf("%d%d", &n, &m);
    for(int i=1;i<n;i++){
        int x, y, z;scanf("%d%d%d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    for(int i=1;i<=m;i++) scanf("%d", &query[i]);
    min_size = inf;
    all = n;
    core = 0;
    getsize(1, 0);
    getsize(core, 0);
    get(core);
    for(int i=1;i<=m;i++) puts(res[i] ? "AYE" : "NAY")
        ;
    return 0;
}
```

2.8 树的重心

//树的重心可以通过简单的两次搜索求出 //实际上这两步操作可以在一次遍历中解决。对节点 u 的每一个儿子 v 递归处理，然后以更新的子节点子树节点数最大值，处理完所有子结点后，判断 u 是否为重心。

```
 void dfs(int u, int pa) { //dfs找重心
   son[u] = 1;
   int res = 0;
   for (int i = head[u]; ~i; i = edge[i].next) {
     int v = edge[i].to;
     if (v == pa) continue;
     dfs(v, u);
     son[u] += son[v];
     res = max(res, son[v]);
   }
   res = max(res, n - son[u]);
   if (res < siz) {
     ans = u;
     siz = res;
   }
 }
 int getCenter(int x) {
   ans = 0;
   siz = INF;
   dfs(x, -1);
   return ans;
 }
```

2.9 树的直径

2.9.1 两次 dfs

复杂度 O(2n)，不能处理负权边，能找出两端位置,bfs 可以记录路径

```
//dfs
typedef ll long long;
const int maxn=1e5+5;
const int INF=0x3f3f3f3f;

struct Edge
{
    int v,l;
    int next;
} edge[maxn<<2];

int vit[maxn],d[maxn];
int head[maxn],k;
int node,ans;

void init()
{
    k=0;
    memset(head,-1,sizeof(head));
}

void addedge(int u,int v,int l)
{
    edge[k].v=v;
    edge[k].l=l;
    edge[k].next=head[u];
    head[u]=k++;

    edge[k].v=u;
    edge[k].l=l;
    edge[k].next=head[v];
    head[v]=k++;
}
void dfs(int u,int t)
{
    for(int i=head[u]; i!=-1; i=edge[i].next)
    {
        int v=edge[i].v;
        if(vit[v]==0)
        {
            vit[v]=1;
            d[v]=t+edge[i].l;
            if(d[v]>ans)
            {
                ans=d[v];
```

```
            node=v;
        }
        dfs(v,d[v]);
    }
  }
}

  memset(vit,0,sizeof(vit)); //加在主函数里
  vit[1]=1;
  ans=0;
  dfs(1,0);

  memset(vit,0,sizeof(vit));
  vit[node]=1;
  ans=0;
  dfs(node,0);

//bfs
const int N = 200010;
vector<pair<int,int> > v[N];
int n, pre[N], vis[N],mark[N];
ll d[N];
int bfs(int s){
  queue<int> q;
  q.push(s);
  memset(vis, 0, sizeof vis);
  memset(d, 0, sizeof d);
  memset(pre, 0, sizeof pre);
  vis[s] = d[s] = 0;
  int t = s;
  while(q.size()){
    int x = q.front();
    q.pop();
    if(d[x] > d[t])***
      t = x;
    vis[x] = 1;
    for (int i = 0; i < v[x].size();i++){
      int y = v[x][i].first;
      if(vis[y])
        continue;
      d[y] = d[x] + v[x][i].second;
      pre[y] = x;****
      q.push(y);
    }
  }
  return t;
}
int main(){
  scanf("%d", &n);
  for (int i = 1,x, y, z; i < n;i++){
    scanf("%d%d%d", &x, &y, &z);
    v[x].push_back({y, z});
    v[y].push_back({x, z});
  }
  int s = bfs(1);
  int t = bfs(s);
  int p = t, np = pre[t];
  ll len = d[t];
  cout << len << endl;
  while(p != s){
    mark[p] = 1;
    p = pre[p];
    mark[p] = 1;
  }
  p = t;
  return 0;
}
```

2.9.2 树形 dp

复杂度 O(n)，能处理负权边，不能找出两端位置

```
void dp(int u)
{
  vis[u] = 1;
  for (int i = head[u]; ~i; i = edge[i].next)
  {
    int v = edge[i].to;
    if(vis[v]) continue;
    dp(v);
    ans = max(ans, d[u] + d[v] + edge[i].w);
    d[u] = max(d[u], d[v] + edge[i].w);
  }
}

    dp(k);//主函数里
    ans=2*sum-ans;//sum为所有边权的和
```

2.10 拓扑排序

```
// poj3687
  #include <iostream>
  #include <cstdio>
  #include <cstdlib>
  #include <cstring>
  using namespace std;

  const int maxn = 205;

  bool g[maxn][maxn];
  bool vis[maxn];
  int in[maxn];
  int ans[maxn];
  int n, m;

  void input()
  {
    scanf("%d%d", &n, &m);
    memset(g, 0, sizeof(g));
    memset(in, 0, sizeof(in));
    for (int i = 0; i < m; i++)
    {
      int a, b;
      scanf("%d%d", &a, &b);
      a--;
      b--;
      if (!g[b][a])
        in[a]++;
      g[b][a] =true;
    }
  }

  void work()
  {
    memset(vis, 0, sizeof(vis));
    for (int i = n; i >0; i--)
    {
      int q =-1;
```

```
        for (int j = n -1; j >=0; j--)
            if (!vis[j] &&in[j] ==0)
            {
                q = j;
                break;
            }
        if (q ==-1)
        {
            printf("-1\n");
            return;
        }
        vis[q] =true;
        ans[q] = i;
        for (int j =0; j < n; j++)
            if (g[q][j])
                in[j]--;
    }
    printf("%d", ans[0]);
    for (int i =1; i < n; i++)
        printf(" %d", ans[i]);
    putchar('\n');
}

int main()
{
int t;
    scanf("%d", &t);
    while (t--)
    {
        input();
        work();
    }
    return 0;
    //system("pause");
}
```

```
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdio>
#include <queue>
using namespace std;
const int maxn = 1e5 + 10;
const int INF = 2147483647;
int n,m ,k, num, root,dcc,c[maxn],uu[maxn*2],vv[
    maxn*2];
int ans, dfn[maxn],low[maxn],du[maxn];
bool bridge[maxn],cut[maxn];
struct Edge
{
    int to;
    int next;
    int w;
};
Edge edge[maxn<<2];
int cnt = 0;
int tot;
int head[maxn], deg[maxn], a[maxn];
void add(int u, int v)
{
    edge[cnt].to = v;
    edge[cnt].next = head[u];
    head[u] = cnt++;
}

void topsort() {
    queue<int> q;
    for(int i = 1; i <= n; i++)
        if(deg[i] == 0) q.push(i);
    while(q.size()) {
        int x = q.front(); q.pop();
        a[++tot] = x;
        for(int i = head[x]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(--deg[v] == 0) q.push(v);
        }
    }
}
int gcd(int a,int b) {return b==0?a:gcd(b,a%b);}
int main(){
    int T,icase = 1;
    scanf("%d", &T);
    while(T--) {
        cin >> n >> m;
        cnt = 0;
        tot = 0;
        memset(head, -1, sizeof(head));
        memset(deg, 0, sizeof(deg));
        for(int i = 1; i <= m; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            add(x, y);
            deg[y]++;
        }
        topsort();
        for (int i = 1; i <= tot; i++) {
            printf("%d ", a[i]);
        }
        cout <<endl;
    }
}
```

2.11 dijkstra

1. 在稠密图: 朴素版 dijkstra O(n2) 2. 在稀疏图: 堆优化版 dijkstra O(mlogn)

2.11.1 朴素算法

```
void Dijkstra(int src){
    for(int i=1; i<=n; ++i)
        d[i] = INF;
    d[src] = 0;
    memset(vis, 0, sizeof(vis));
    for(int i=1; i<=n; ++i){
        int u=-1;
        for(int j=1; j<=n; ++j)if(!vis[j]){
            if(u==-1 || d[j]<d[u]) u=j;
        }
        vis[u] = 1;
        for(int j=1; j<=n; ++j)if(!vis[j]){
            int tmp = d[u] + w[u][j];
            if(tmp<d[j]) d[j] = tmp;
        }
    }
}
```

2.11.2 堆优化

```
void Dijkstra(int src){
    for(int i=1; i<=n; ++i)
        d[i] = INF;
    d[src] = 0;
    memset(vis, 0, sizeof(vis));
    for(int i=1; i<=n; ++i){
        int u=-1;
        for(int j=1; j<=n; ++j)if(!vis[j]){
            if(u==-1 || d[j]<d[u]) u=j;
        }
        vis[u] = 1;
        for(int j=1; j<=n; ++j)if(!vis[j]){
            int tmp = d[u] + w[u][j];
            if(tmp<d[j]) d[j] = tmp;
        }
    }
}
```

2.12 SPFA

2.12.1 求最短路

```
void spfa(int u)
{
    memset(d, 0x3f, sizeof(d)), memset(vis, 0, sizeof(
        vis));
    d[u] = 0,vis[u] = 1, q.push(u);
    while(!q.empty()){
        int x = q.front();q.pop();
        vis[x] = 0;
        for(int i = head[u]; ~i; i = edge[i].next){
            int y = edge[i].to, z =edge[i].w;
            if(d[y] > d[x] + z){
                d[y] = d[x] + z;
                if(!vis[y]) q.push(y), vis[y] = 1;
            }
        }
    }
}
```

2.12.2 判断负环

```
//设res(x)表示从1~x的最短路径包含的边数，若发现res(x)≥n
    ，则图中有负环。
bool spfa()
{
    memset(d, 0x3f, sizeof(d)),memset(vis,0,sizeof(vis
        ));
    d[1] = 0,vis[1] = 1, res[1] = 0;
        q.push(1);
    while(!q.empty()){
        int x = q.front();q.pop();
        vis[x] = 0;
        for(int i = head[x]; ~i; i = edge[i].next){
            int y = edge[i].to, z =edge[i].w;
            if(d[y] > d[x] + z){
                d[y] = d[x] + z;
                res[y] = res[x] + 1;
                if(res[y] >= n) return true;
                if(!vis[y]){
                        q.push(y), vis[y] = 1;
                }

            }
        }
    }
    return false;
}
```

2.12.3 差分约束

```
bool spfa()
{
    for(int i = 0; i <= n; i++){
        d[i] = INF;
        vis[i] = 0;
    }
    d[1] = 0,vis[1] = 1,res[1] = 0;
        q.push(1);
    while(!q.empty()){
        int x = q.front();q.pop();
        vis[x] = 0;
        for(int i = head[x]; ~i; i = edge[i].next){
            int y = edge[i].to, z =edge[i].w;
            if(d[y] > d[x] + z){ //要求最长的距离应求最短
                路
                d[y] = d[x] + z;
                res[y] = res[x] + 1;
                if(!vis[y]){
                    if(res[y] >= n) return true;
                    q.push(y), vis[y] = 1;
                }
            }
        }
    }
    return false;
}

    for(int i = 1; i <= m; i++){ //距离不大于
        scanf("%d%d%d", &u, &v, &w);
        add(u,v,w);
    }
    for(int i = 1; i <= k; i++){ //距离不小于
        scanf("%d%d%d", &u, &v, &w);
        add(v,u,-w);
```

2.13 Floyd

2.13.1 求关系闭包（递闭包）

```
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdio>
#include <queue>
#include <set>
using namespace std;
const int maxn = 1e4 + 10;
const int INF = 2147483647;
int n, t, m;
```

```
int ans, sum, d[maxn][maxn];  //Floyd一般用邻接矩阵存图
int main()
{
    cin>>n>>m;
    int u, v;
    for(int i = 1; i <= n; i++)
        for(int j=1; j<= n;j++)
            d[i][j]=INF;
    for(int i = 1; i <= m; i++){
            scanf("%d%d", &u, &v);
            d[u][v] = 1; //u胜v为1
            d[v][u] = -1; //v负u为-1
    }
    for(int k = 1;k <= n; k++)
        for(int i = 1; i <= n; i++)
            for(int j=1; j <= n; j++){
                if(d[i][k] == d[k][j] && (d[i][k] == 1
                    || d[i][k] == -1)) //求关系闭包，若
                    //若A→B，B→C，则A→C
                    d[i][j] = d[i][k];
            }
    for(int i = 1; i<= n; i++){
        sum = 0;
        for(int j = 1; j <= n; j++){
            if(d[i][j]!=INF) sum++;
        }
        if(sum == n-1) ans++;
    }
    cout << ans << endl;
    //system("pause");
    return 0;
}
```

2.13.2 多源最短路

```
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdio>
#include <queue>
#include <set>
using namespace std;
const int maxn = 1e4 + 10;
const int INF = 2147483647;
int n, t, m;
int ans, sum, d[maxn][maxn];  //Floyd一般用邻接矩阵存图
int main()
{
    cin>>n>>m;
    int u, v;
    for(int i = 1; i <= n; i++)
        for(int j=1; j<= n;j++)
            d[i][j]=INF;
    for(int i = 1; i <= m; i++){
            scanf("%d%d", &u, &v);
            d[u][v] = 1; //u胜v为1
            d[v][u] = -1; //v负u为-1
    }
    for(int k = 1;k <= n; k++)
        for(int i = 1; i <= n; i++)
            for(int j=1; j <= n; j++){
                if(d[i][k] == d[k][j] && (d[i][k] == 1
                    || d[i][k] == -1)) //求关系闭包，若
                    //若A→B，B→C，则A→C
                    d[i][j] = d[i][k];
            }
    for(int i = 1; i<= n; i++){
        sum = 0;
        for(int j = 1; j <= n; j++){
            if(d[i][j]!=INF) sum++;
        }
        if(sum == n-1) ans++;
    }
    cout << ans << endl;
    //system("pause");
    return 0;
}
```

2.14 LCA(最近公共祖先)

```
    /*- **定义**: 两个节点的最近公共祖先，就是这两个点的公共祖先里面，离根最远的那个。

    - **性质**: 1.两点最近公共祖先必在树上两点间最短路上

        2.d是树上两点间距离，h为点到树根的距离
      $$
      d(u,v) = h(u) + h(v) -2h(LCA(u,v))
      $$

    **Tarjan算法**

    - 使用并查集对“向上标记法”的优化
    - 三类节点: 1.vis[x] == 2 已访问并回溯
        2.vis[x] == 1 已开始递归但未回溯，正在访问的节点x以及x的祖先
        3.vis[x] == 0 尚未访问的节点*/
int getf(int x) //寻找父亲
{
    return f[x] == x ? x : f[x] = getf(f[x]);
}

void Tarjan(int u)
{
    vis[u] = 1;
    int to;
    for (int i = head[u]; ~i; i = edge[i].next)
    {
        to = edge[i].to;
        if (vis[to]) continue;
        d[to] = d[u] + edge[i].w;
        Tarjan(to);
        f[to] = u;
    }
    for (int i = 0; i < query[u].size(); i++)
    {
        int to = query[u][i],id = quert_id[u][i];
        if (vis[to] == 2)
        {
            int lca = getf(to);
            ans[id] = min(ans[id], d[u] + d[to] - 2 * d[lca]);
        }
```

```
    }
    vis[u] = 2; //一回溯，标记为2
}

/*小技巧：存储离线一次性读入
$$
vector<int> query[N]
$$
*/

vector<int> query[maxn], quert_id[maxn];
void add_query(int x, int y, int id) //只要有关，就要加入询问
{
    query[x].push_back(y), quert_id[x].push_back(id);
    query[y].push_back(x), quert_id[y].push_back(id);
}
for (int i = 1; i <= m; ++i) //读入询问！！！学会方法
{
    int x, y;
    scanf("%d%d", &x, &y);
    if(x == y) ans[i] = 0;
    else {
    add_query(x,y,i);
    ans[i] = 1 << 30;
    }
}
```

2.15 二分图匹配（染色法）

```
    /*定理：一张图是二分图，当且仅当图中不存在奇环(长度为奇数的环)。
      方法：121212…染色，没有矛盾就是二分图
    */
void dfs(int u, int c)
{
    color[u] = c;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(!color[v]){
            dfs(v,3-c);
        }
    }
}
int main()
{
    cin >> t;
    while(t--){
        scanf("%d%d", &n,&m);
        for(int i = 1; i <= n; i++)
        color[i]=0,head[i]=-1; //不要用memset，会超时！！！
        cnt = 0;
        int u, v;
        for(int i = 0; i < m; i++){
            scanf("%d%d",&u,&v);
            add(u,v);
            add(v,u);
        }
        dfs(1,1);
        k = 0, l = 0;
        for(int i = 1; i <= n; i++){
            if(color[i] == 1) a[++k] = i;
            else b[++l] = i;
        }
        bool flag;
        if(k <= n/2) flag = 1;
        else flag = 0;
        if(flag){
            printf("%d\n",k);
            for(int i = 1; i <=k; i++) printf("%d ",a[i]);
            cout<<endl;
        }
        else
        {
            printf("%d\n",l);
            for(int i = 1; i <=l; i++) printf("%d ",b[i]);
            cout<<endl;
        }
    }
    //system("pause");
    return 0;
}

//也可用用bfs
bool bfs1(int s)
{
    queue<int> q;
    color[s] = 1;
    q.push(s);
    while(q.size()){
        int u = q.front();
        q.pop();
        for(int i = head[u]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(!color[v]){
                color[v] = 3-color[u];
                q.push(v);
            }
            else if(color[v] == color[u] ) return false;
        }
    }
    return true;
}
```

2.16 二分图最大匹配（匈牙利算法）

```
/*最大匹配：“任意两条边都没有公共端点”的边的集合被称为图的一组**匹配**，在二分图中，包含边最 多的一组匹配被称为二分图的最大匹配。5

  寻找增广路：1.y本身是非匹配点
            2.y已与左部点x1匹配，但从x1出发可找到另一个右部点y1匹配，此时x~y~x1~y1是一条增广路
*/
bool dfs2(int u)
{
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(!vis[v]){
            vis[v] = 1;
            if(!match[v] || dfs2(match[v])){
                match[v] = u;
                return true;
            }
```

```
        }
    }
    return false;
}
for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++) vis[j] = 0;
        if(dfs2(i)) ans++;
}
```

2.17 Tarjan 求割边

```
/*时间戳: 在图的深度优先遍历过程中，按照每个节点第一次被访问的时间顺序，依次给予1~n的数字标记，该标记被称为”时间戳”。***dfn[i]***表示

subtree(x)表示搜索树中以x为根的子树结点的集合。

low(x)（追溯值）**表示顶点x及其子树中的点，通过非搜索树上的边，能够回溯到的最早的点(dfn最小)的dfn值(但不能连接x与其父节点的边)。

割边判定法则：dfn(x)<low(y)
*/
void tarjan1(int u, int in_edge)
{
    dfn[u] = low[u] = ++num;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(!dfn[v]){
            tarjan1(v, i);
            low[u] = min(low[v], low[u]);
            if(low[v] > dfn[u]){
                bridge[i] = bridge[i ^1] = true;
            }
        }
        else if(i != (in_edge ^ 1) )
            low[u] = min (low[u],dfn[v]);
    }
}
```

2.18 Tarjan 求割点

```
    void tarjan2(int u)
    {
        dfn[u] = low[u] = ++num;
        int flag = 0;
        for(int i = head[u]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(!dfn[v]){
                tarjan2(v);
                low[u] = min(low[v], low[u]);
                if(low[v] >= dfn[u]){
                    flag++;
                    if(flag > 1 || u != root)cut[u] =
                        true;
                }
            }
            else low[u] = min (low[u],dfn[v]);
        }
    }

//加边中
if(u == v) continue;
//主函数中
for(int i = 1;i <= n; i++){
    if(!dfn[i]) root = i, tarjan2(i);
}
for(int i = 1;i <= n; i++){
    if(cut[i]) printf("%d",i);
}
```

2.19 网络流

2.19.1 最大流

整个网络流量最大，流函数 f 为最大流
点上有权值，都要拆点

```
//**增广路算法**
/*O(nm2) 一般可以处理103□104 规模的网络
- 各条边剩余流量大于0，为一条增广路，不断用/bfs求增广路

- O(nm^2),1000~10000
- 为了存反向边，邻接表成对存储
- （x,y）减小e，(y,x)增大e
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1010, M = 10010, inf = 1 << 29;
int head[N], next[N], ver[M], edge[M], v[N], incf[N],
    pre[N];
int n, m, s, t, tot, maxflow;
void add(int x, int y, int z){
    ver[++tot] = y; edge[tot] = z; next[tot] = head[x
        ]; head[x] = tot;
    ver[++tot] = x; edge[tot] = 0; next[tot] = head[y
        ]; head[y] = tot;
}

bool bfs(){
    memset(v,0,sizeof(v));
    queue<int> q;
    q.push(s), v[s] = 1;
    incf[s] = inf; //增广路各边最小剩余流量
    while(q.size()){
        int x = q.front();
        q.pop();
        for(int i = head[x]; i; i = next[i]){
            if(edge[i]){
                int y = ver[i];
                if(v[y]) continue;
                incf[y] = min(incf[x], edge[i]); //最小
                    剩余容量作为此条增广路流量
                pre[y] = i;
                q.push(y), v[y] = 1;
                if(y == t) return 1;
            }
        }
    }
    return 0;
}

void update(){//更新增广路和反向边剩余流量
    int x = t;
    while(x != s){
        int i = pre[x];
```

```
        edge[i] -= incf[t];
        edge[i^1] += incf[t]; // “成对存储"xor 1 ,找到反向的容量
        x = ver[i^1];
    }
    maxflow += incf[t];
}

int main(){
    while(cin >> n >> m){
        memset(head, 0, sizeof(head));
        s = 1; t = n; tot = 1; maxflow = 0;
        for(int i = 0; i < m; i++){
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            add(x, y ,z);
        }
        while(bfs()) update();
        cout << maxflow << endl;
    }
}
/*
7 10
1 2 4
1 3 5
2 3 1
2 4 5
3 4 2
3 5 5
4 7 5
5 4 6
5 6 3
6 7 2
*/
```

Dinic 算法 **（更快，1e4 1e5）1. BFS 求出节点层次，构造分层图 2. 分层图 DFS 找增广路，回溯时更新剩余容量，每个点可以流向多条边，还要剪枝

```
//O(n2m) 该算法求解二分图最大匹配的时间复杂度为O(mn--√)
    #include <bits/stdc++.h>
    using namespace std;
    typedef long long ll;
    const int N = 5010, M = 30010, inf = 1 << 29;
    int n,m,s,t,tot,maxflow;
    int ver[M], next[M], head[N], edge[M], d[N];
    queue<int> q;
    void add(int x, int y, int z){
        ver[++tot] = y; edge[tot] = z; next[tot] = head[x]; head[x] = tot;
        ver[++tot] = x; edge[tot] = 0; next[tot] = head[y]; head[y] = tot;
    }

    bool bfs(){
        memset(d,0,sizeof(d));
        while(q.size()) q.pop(); //每一次生成分层图之前先清空栈
        q.push(s), d[s] = 1;
        while(q.size()){
            int x = q.front(); q.pop();
            for(int i = head[x]; i; i = next[i]){
                if(edge[i] && !d[ver[i]]){//d[ver[i]](与x相连的下一个点）不为零，说明已构成了分层图中的一层，不满足条件
                    d[ver[i]] = d[x] + 1;
                    q.push(ver[i]);
                    //cout << "1" <<endl;
                    if(ver[i] == t) return 1;//可以到达汇点，说明可以构成一个分层图
                }
            }
        }
        return 0;
    }

    int dinic(int x, int flow){
        if(x == t) return flow;
        int rest = flow, k;
        for(int i = head[x]; i && rest; i = next[i]) //rest为什么不能等于0
            if(edge[i] && d[ver[i]] == d[x] + 1){
                k = dinic(ver[i], min(edge[i], rest));
                if(!k) d[ver[i]] = 0; //剪枝，去掉增广完毕的点(什么意思？)
                edge[i] -= k;
                edge[i^1] += k;
                rest -= k;
            }
        return flow - rest; //相当于返回 k（已经找到的流流量）
    }

    int main(){
        while(cin >> n >> m){
            s = 1; t = n; tot = 1; maxflow = 0;
            for(int i = 0; i < m; i++){
                int x, y, z;
                scanf("%d%d%d", &x, &y, &z);
                add(x, y ,z);
            }
            int flow = 0;
            while(bfs())
                while(flow = dinic(s, inf)) maxflow += flow;
            cout << maxflow <<endl;
        }
        system("pause");
    }
```

2.19.2 最小割

```
// poj 1966
// 题意：无向图，去掉点，使不连通
//技巧：点边转化
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 5010, M = 30010, inf = 1e9;
int n,m,s,t,tot,maxflow,ans;
int ver[M], Next[M], head[N], edge[M], d[N];
queue<int> q;
int x[M], y[M];

void add(int x, int y, int z){
    ver[++tot] = y; edge[tot] = z; Next[tot] = head[x]; head[x] = tot;
    ver[++tot] = x; edge[tot] = 0; Next[tot] = head[y]; head[y] = tot;
}
```

```

bool bfs(){
    memset(d,0,sizeof(d));
    while(q.size()) q.pop(); //每一次生成分层图之前先清空栈
    q.push(s), d[s] = 1;
    while(q.size()){
        int x = q.front(); q.pop();
        for(int i = head[x]; i; i = Next[i]){
            if(edge[i] && !d[ver[i]]){ //d[ver[i]] (与x相连的下一个点) 不为零，说明已构成了分层图中的一层，不满足条件
                d[ver[i]] = d[x] + 1;
                q.push(ver[i]);
                    //cout << "1" <<endl;
                if(ver[i] == t) return 1;//可以到达汇点，说明可以构成一个分层图
            }
        }
    }
    return 0;
}

int dinic(int x, int flow){
    if(x == t) return flow;
    int rest = flow, k;
    for(int i = head[x]; i && rest; i = Next[i]) //rest为什么不能等于0
        if(edge[i] && d[ver[i]] == d[x] + 1){
            k = dinic(ver[i], min(edge[i], rest));
            if(!k) d[ver[i]] = 0; //剪枝，去掉增广完毕的点(什么意思？)
            edge[i] -= k;
            edge[i^1] += k;
            rest -= k;
        }
    return flow - rest; //相当于返回 k (已经找到的流流量)
}

int main(){
    while(cin >> n >> m){
        char str[100];
        for(int i=1;i<=m;++i)
        {
            scanf("%s",&str); int j;
            for(x[i]=0,j=1;str[j]!=',';j++) x[i]=x[i]*10+str[j]-'0';
            for(y[i]=0,j++;str[j]!=')';j++) y[i]=y[i]*10+str[j]-'0';
            x[i]++; y[i]++;
        }
        ans = inf;
        for(s = 1; s <= n; s++)
            for(t = 1; t <= n; t++){
                if(s!=t){
                    memset(head, 0, sizeof(head));
                    tot = 1; maxflow = 0;
                    int flow;
                    for(int i = 1; i <=n ; i++) {
                        if(s == i || t == i) add(i, i+n, inf);
                        else add(i, i+n, 1);
                    }
                    for(int i = 1; i <= m; i++){
                        add(x[i]+n, y[i], inf);
                        add(y[i]+n, x[i], inf);
                    }
                    while(bfs())
                        while(flow = dinic(s, inf))
                            maxflow += flow;
                    ans = min(ans, maxflow);
                }
        }
        if(ans == inf||n<=1) ans = n;
        cout << ans << endl;
    }
}
```

2.19.3 有向无环图最小路径覆盖

尽量少的不相交的简单路径，每个顶点恰好覆盖一次

拆点，x, x+n, 1 n 为左部点，n+1 2n 为右部点，得到拆点二分图

定理：最小路径点覆盖包含路径点条数 = n - 拆点二分图最大匹配数

```
/*
libre oj 6002 最小路径覆盖

**记录路径方法**:  用一个nex[]数组，在增广成功时，nex[x] = to - n(下一个点)；然后遍历起点（没有进入的点，）
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 5010, M = 30010, inf = 1e9;
int n,m,s,t,tot,maxflow,ans;
int ver[M], Next[M], head[N], edge[M], d[N];
queue<int> q;
int nex[N];

void add(int x, int y, int z){
    ver[++tot] = y; edge[tot] = z; Next[tot] = head[x]; head[x] = tot;
    ver[++tot] = x; edge[tot] = 0; Next[tot] = head[y]; head[y] = tot;
}

bool bfs(){
    memset(d,0,sizeof(d));
    while(q.size()) q.pop(); //每一次生成分层图之前先清空栈
    q.push(s), d[s] = 1;
    while(q.size()){
        int x = q.front(); q.pop();
        for(int i = head[x]; i; i = Next[i]){
            if(edge[i] && !d[ver[i]]){ //d[ver[i]] (与x相连的下一个点) 不为零，说明已构成了分层图中的一层，不满足条件
                d[ver[i]] = d[x] + 1;
                q.push(ver[i]);
                if(ver[i] == t) return 1;//可以到达汇点，说明可以构成一个分层图
            }
        }
    }
    return 0;
}

int main(){
    while(cin >> n >> m){
        char str[100];
        for(int i=1;i<=m;++i)
        {
            scanf("%s",&str); int j;
            for(x[i]=0,j=1;str[j]!=',';j++) x[i]=x[i]*10+str[j]-'0';
            for(y[i]=0,j++;str[j]!=')';j++) y[i]=y[i]*10+str[j]-'0';
            x[i]++; y[i]++;
        }
        ans = inf;
        for(s = 1; s <= n; s++)
            for(t = 1; t <= n; t++){
                if(s!=t){
                    memset(head, 0, sizeof(head));
                    tot = 1; maxflow = 0;
                    int flow;
                    for(int i = 1; i <=n ; i++) {
                        if(s == i || t == i) add(i, i+n, inf);
                        else add(i, i+n, 1);
                    }
                    for(int i = 1; i <= m; i++){
                        add(x[i]+n, y[i], inf);
                        add(y[i]+n, x[i], inf);
                    }
                    while(bfs())
                        while(flow = dinic(s, inf))
                            maxflow += flow;
                    ans = min(ans, maxflow);
                }
        }
        if(ans == inf||n<=1) ans = n;
        cout << ans << endl;
    }
}
```

```
int dinic(int x, int flow){
    if(x == t) return flow;
    int rest = flow, k;
    for(int i = head[x]; i && rest; i = Next[i]) //
        rest为什么不能等于0
        if(edge[i] && d[ver[i]] == d[x] + 1){
            k = dinic(ver[i], min(edge[i], rest));
            if(!k) d[ver[i]] = 0; //剪枝，去掉增广完毕的
                点(什么意思？)
            edge[i] -= k;
            edge[i^1] += k;
            rest -= k;
            if(k) nex[x] = ver[i] - n;
        }
    return flow - rest; //相当于返回 k（已经找到的流流
        量)
}

int main(){
    while(cin >> n >> m){
        s = 0, t = n + 1, ans = 0 , tot = 1 , maxflow
            = 0;
        for(int i=1;i<=n;++i) {
            add(s, i, 1); add(i+n, t, 1);
        }
        for(int i = 1; i <= m; i++) {
            int x, y;
            scanf("%d%d", &x,&y);
            add(x, y+n, 1);
        }
        int flow = 0;
        while(bfs())
            while(flow = dinic(s, inf)) maxflow += flow
                ;
        for(int i = head[t]; i; i = Next[i]) {
            if(edge[i^1]){
                int p = ver[i] - n;
                while(p) printf("%d ", p), p = nex[p];
                puts("");
            }
        }
        ans = n - maxflow;
        cout << ans << endl;
    }
}
```

3 数据结构

3.1 并查集

```
int n,m,x,y;
int f[N],ronk[N];

void init()//建立森林
{
    for(int i=0;i<n;i++)
    {
        f[i]=i;ronk[i]=0;
    }
}


int getf(int x){
    return f[x]==x? x:f[x]=getf(f[x]);
}

void unionset(int i,int j){
    x=getf(i);
    y=getf(j);
    if(x==y) return;
    else{
        f[x]=y;
    }
}

void unionset(int i,int j)//合并并查集，根据秩的大小，
    小的连在大的上面
{
    x=getf(i);
    y=getf(j);
    if(x==y) return;
    if(ronk[x]>ronk[y])
        f[y]=x;
    else
    {
        f[x]=y;
        if(ronk[x]==ronk[y])
            ronk[x]++;
    }
}

for (int i=1;i<=n;i++) num[getf(i)]++;//求元素所在集合
    元素个数
for (int i=1;i<=n;i++) cout<<num[getf(i)]<<' ';

void unionset(int i,int j)//求个数时误用，会超时！！！
{
    int x=getf(i);
    int y=getf(j);
    if(x==y) return;
    if(ronk[x]>ronk[y]){
        f[y]=x;
        num[x]+=num[y];
    }
    else
    {
        f[x]=y;
        if(ronk[x]==ronk[y])
            ronk[x]++;
            num[y]+=num[x];
    }
}
```

带权并查集

```
#include <bits/stdc++.h>
using namespace std;
const int N = 4000010;
int fa[N],to[N],sz[N],cnt,a[N],b[N],tot;
int n,m,k,x,y;
int ok[N];
int find(int x){
    return x == fa[x]?x:fa[x] = find(fa[x]);
}
void merge(int x,int y){
    int a = find(to[x]);
    int b = find(to[y]);
    if(a==b) return;
    fa[b] = a;
    sz[a] += sz[b]; sz[b] = 0;
```

```
}
void update(int x,int y){
    sz[find(to[x])]--;
    to[x] = ++cnt;
    sz[cnt] = 1;
    fa[cnt] = cnt;
    merge(x,y);
}
int main(){
    scanf("%d%d",&n,&m);cnt = n;
    for(int i=1;i<=n;i++)fa[i] = i,to[i] = i,sz[i] = 1;
    for(int i=1;i<=m;i++){
        scanf("%d%d",&k,&x);if(k!=3)scanf("%d",&y);
        if(k == 5){
            a[++tot] = x;
            b[tot] = y;continue;
        }
        if(k == 1)merge(x,y);
        if(k == 2)update(x,y);
        if(k == 4){
            if(find(to[x]) == find(to[y]))printf("Yes\n");
            else printf("No\n");
        }

        if(k == 3)printf("%d\n",sz[find(to[x])]-1);
    }

    for(int i=1;i<=tot;i++){
        if(find(to[a[i]]) == find(to[b[i]])) ok[find(to[a[i]])] = 1;
    }
    int res = -1;
    for(int i=1;i<=n;i++)
        if(!ok[find(to[i])])res = max(res,sz[find(to[i])]);
    cout<<res<<endl;
    return 0;
}
```

可持久化并查集

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int inf = 0x3f3f3f3f;
#define dbg(x...) do { cout << "\033[32;1m" << #x <<" -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<class T, class... Ts> void err(const T& arg, const Ts&... args) { cout << arg << " "; err(args...); }
const int N = 200000 + 5;

int n, m;
struct TreeNode{
    int l, r;
};
int root[N];
class Union_Find{
public:
    int fa[N*30], dep[N*30], tot;
    TreeNode t[N*30];
    void build(int &p, int l, int r){
        p = ++tot;
        if(l == r){ fa[p] = l; return; }
        int mid = l + r >> 1;
        build(t[p].l, l, mid);
        build(t[p].r, mid+1, r);
    }
    void change(int last, int &p, int l, int r, int pos, int val){
        p = ++tot;
        t[p].l = t[last].l, t[p].r = t[last].r;
        if(l == r){
            fa[p] = val;
            dep[p] = dep[last];
            return;
        }
        int mid = (l + r) >> 1;
        if(pos <= mid) change(t[last].l, t[p].l, l, mid, pos, val);
        else change(t[last].r, t[p].r, mid + 1, r, pos, val);
    }
    int getIndex(int p, int l, int r, int pos){
        if(l == r) return p;
        int mid = l + r >> 1;
        if(pos <= mid) return getIndex(t[p].l, l, mid, pos);
        else return getIndex(t[p].r, mid+1, r, pos);
    }
    void add(int p, int l, int r, int pos){
        if(l == r){
            dep[p] ++;
            return;
        }
        int mid = l + r >> 1;
        if(pos <= mid) add(t[p].l, l, mid, pos);
        else add(t[p].r, mid+1, r, pos);
    }
    // find 返回的是祖先节点的下标
    int find(int p, int pos){
        int index = getIndex(p, 1, n, pos);
        if(fa[index] == pos)
            return index;
        else
            return find(p, fa[index]);
    }
    void merge(int last, int &p, int x, int y){
        int fax = find(p, x), fay = find(p, y);
        if(fa[fax] == fa[fay]) return;
        if(dep[fax] > dep[fay]) swap(fax, fay);
        change(last, p, 1, n, fa[fax], fa[fay]);
        if(dep[fax] == dep[fay]) add(p, 1, n, fa[fay]);
    }
}uf;

int main(){
    scanf("%d%d", &n, &m);
    uf.build(root[0], 1, n);
    for(int i=1;i<=m;i++){
        int op, x, y;
        scanf("%d%d", &op, &x);
        if(op == 1){
            scanf("%d", &y);
            root[i] = root[i-1];
            uf.merge(root[i-1], root[i], x, y);
        } else if(op == 2) root[i] = root[x];
```

```
        else{
            scanf("%d", &y);
            root[i] = root[i-1];
            int fax = uf.find(root[i], x);
            int fay = uf.find(root[i], y);
            if(uf.fa[fax] == uf.fa[fay]) puts("1");
            else puts("0");
        }
    }

    return 0;
}
```

3.2 分块

```
int n,m,x,y;
int a[N], sum[N], add[N], pos[N], L[N], R[N];//sum是块内所有元素的和 add是每一块一致变化的更改值 pos是当前元素属于那一块

void create(ll n) { //分块
    ll t = sqrt(n);
    for(ll i = 1; i <= t; i++) {
        L[i] = (i-1) * sqrt(n) + 1;
        R[i] = i*sqrt(n);
    }
    if(R[t] < n) t++, L[t] = R[t-1] + 1, R[t] = n;
    for(ll i = 1; i <= t; i++)
        for(ll j = L[i]; j <= R[i]; j++) {
        pos[j] = i;
        sum[i] += a[j];
    }
}

void update(ll l, ll r, ll d) { //更新
    ll p = pos[l], q = pos[r];
    if(p == q) {
        for(ll i = l; i <= r; i++) a[i] += d;
        sum[p] += d*(r-l+1);
    }
    else {
        for(ll i = p + 1; i < q; i++) {
            add[i] += d;
        }
        for(ll i = l; i <= R[p]; i++) {
            a[i] += d;
            sum[p] += d;
        }
        for(ll i = L[p]; i <= r; i++) {
            a[i] += d;
            sum[q] += d;
        }
    }
}

ll ask(ll l, ll r) { //询问区间和
    ll ans = 0;
    ll p = pos[l], q = pos[r];
    if(p == q) {
        for(ll i = l; i <= r; i++) ans += a[i] + add[p];
    }
    else {
        for(ll i = p + 1; i < q; i++)
            ans += sum[i] + add[i] * (R[i] - L[i] + 1);
        for(ll i = l; i <= R[p]; i++)
            ans += a[i] + add[p] * (R[i] - l + 1);
        for(ll i = L[q]; i <= r; i++)
            ans += a[i] + add[p] * (r -L[i] + 1);
    }
    return ans;
}
```

3.3 树状数组

```
int lowbit(int x) {
    return x & -x;
}
void init(){//线性构造
    for (int i = 1; i <= n; i++){
        pre[i] = pre[i - 1] + a[i];
        c[i] = pre[i] - pre[i - lowbit(i)];
    }
}

int ask(int x) { //查询前缀和 区间和[l,r]:ask(r) - ask(l-1);
    int ans = 0;
    for(; x; x -= x & -x) ans += c[x];
    return res;
}

void update(int x, int t) { //单点增加
    while(x <= n) {
        c[x] += t;
        x += lowbit(x);
    }
}
```

3.4 线段树

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
#include <vector>
using namespace std;
const int size = 1e5+10;
int a[size];
struct SetmentTree
{
    int l, r, dat;
}t[size * 4];

void BuildTree(int p, int l, int r) //线段树建树
{
    t[p].l = l, t[p].r = r;
    if(l == r){
        t[p].dat = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    BuildTree(p * 2, l, mid),BuildTree(p * 2 + 1, mid + 1, r);
```

```
    t[p].dat = max(t[p * 2].dat, t[p * 2 + 1].dat);
}

void change(int p, int x, int v){ //单点修改
    if(t[p].l == t[p].r){
        t[p].dat = v;
        return;
    }
    int mid = (t[p].l + t[p].r) >> 1;
    if(x <= mid) change(p * 2, x, v);
    else change(p * 2 + 1, x, v);
    t[p].dat = max(t[p*2].dat, t[p*2+1].dat);
}

int ask(int p, int l, int r){ //区间查询
    if(l <= t[p].l && r >= t[p].r) return t[p].dat;
    int mid = (t[p].l + t[p].r) >> 1;
    int val = -(1 >> 30);
    if(l <= mid) val = max(val, ask(p*2, l, r));
    if(r > mid) val = max(val, ask(p*2+1, l, r));
    return val;
}
int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        scanf("%d",&a[i]);
    }
    cout<<"111"<<endl;
    BuildTree(1, 1, n);
    int T;
    scanf("%d",&T);
    while(T--){
        int x, v;
        scanf("%d%d",&x,&v);
        change(1, x, v);
        cout << t[1].dat <<endl;
    }
    cout << ask(1, 1 ,3);
        system("pause");

}

//建树

void build(int s, int t, int p) {
  // 对 [s,t] 区间建立线段树,当前根的编号为 p
  if (s == t) {
    d[p] = a[s];
    return;
  }
  int m = (s + t) / 2;
  build(s, m, p * 2), build(m + 1, t, p * 2 + 1);
  // 递归对左右区间建树
  d[p] = d[p * 2] + d[(p * 2) + 1];
}

// 区间查询

//区间查询，比如求区间 的总和（即 ）、求区间最大值/最小值等操作。

int getsum(int l, int r, int s, int t, int p) {
  // [l,r] 为查询区间,[s,t] 为当前节点包含的区间,p 为当前节点的编号
  if (l <= s && t <= r)
    return d[p]; // 当前区间为询问区间的子集时直接返回当前区间的和
  int m = (s + t) / 2, sum = 0;
  if (l <= m) sum += getsum(l, r, s, m, p * 2);
  // 如果左儿子代表的区间 [l,m] 与询问区间有交集,则递归查询左儿子
  if (r > m) sum += getsum(l, r, m + 1, t, p * 2 + 1);
  // 如果右儿子代表的区间 [m+1,r] 与询问区间有交集,则递归查询右儿子
  return sum;
}

//区间修改与懒惰标记

//区间修改（区间加上某个值）：

void update(int l, int r, int c, int s, int t, int p) {
  // [l,r] 为修改区间,c 为被修改的元素的变化量,[s,t] 为当前节点包含的区间,p
  // 为当前节点的编号
  if (l <= s && t <= r) {
    d[p] += (t - s + 1) * c, b[p] += c;
    return;
  } // 当前区间为修改区间的子集时直接修改当前节点的值,然后打标记,结束修改
  int m = (s + t) / 2;
  if (b[p] && s != t) {
    // 如果当前节点的懒标记非空,则更新当前节点两个子节点的值和懒标记值
    d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[p] * (t - m);
    b[p * 2] += b[p], b[p * 2 + 1] += b[p]; // 将标记下传给子节点
    b[p] = 0; // 清空当前节点的标记
  }
  if (l <= m) update(l, r, c, s, m, p * 2);
  if (r > m) update(l, r, c, m + 1, t, p * 2 + 1);
  d[p] = d[p * 2] + d[p * 2 + 1];
}

//区间查询（区间求和）：

int getsum(int l, int r, int s, int t, int p) {
  // [l,r] 为修改区间,c 为被修改的元素的变化量,[s,t] 为当前节点包含的区间,p
  // 为当前节点的编号
  if (l <= s && t <= r) return d[p];
  // 当前区间为询问区间的子集时直接返回当前区间的和
  int m = (s + t) / 2;
  if (b[p]) {
    // 如果当前节点的懒标记非空,则更新当前节点两个子节点的值和懒标记值
    d[p * 2] += b[p] * (m - s + 1), d[p * 2 + 1] += b[p] * (t - m),
    b[p * 2] += b[p], b[p * 2 + 1] += b[p]; // 将标记下传给子节点
    b[p] = 0; // 清空当前节点的标记
  }
  int sum = 0;
  if (l <= m) sum = getsum(l, r, s, m, p * 2);
  if (r > m) sum += getsum(l, r, m + 1, t, p * 2 + 1)
```

```
    ;
  return sum;
}
```

3.5 bfs

3.5.1 小猫爬山

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e2;
int c[maxn], cab[maxn], cnt, ans, n, w;

bool cmp(int a, int b){
    return a > b;
}

void dfs(int now, int cnt){
    if(cnt >= ans) return;
    if(now == n + 1){
        ans = min(cnt, ans);
        return;
    }
    for(int i = 1; i <= cnt; i++){
        if(cab[i] + c[now] <= w){
            cab[i] += c[now];
            dfs(now + 1, cnt);
            cab[i] -= c[now]; //还原现场（为了尝试将小猫放在下一个缆车里）
        }
    }
    cab[cnt + 1] = c[now];
    dfs(now + 1, cnt + 1);
    cab[cnt + 1] = 0; //还原现场（不将这个猫放在这个地方）
}

int main()
{
    while(cin >> n >> w){
        for(int i = 1; i <= n; i++) cin >> c[i];
        sort(c + 1, c + n + 1, cmp);
        ans = n;
        dfs(1, 0);
        cout << ans << endl;
    }
    //system("pause");
    return 0;
}
```

3.5.2 八皇后

```
int n,x;
char g[N][N];
int col[N], dg[N], udg[N];
void dfs(int p) {
    if(p == n) {
        for(int i = 0; i < n; i++) {
            puts(g[i]);
        }
        puts("");
        return;
    }
    for(int i = 0; i < n; i++){
        if(!col[i] && !dg[i+p] && !udg[n+i-p]) {
            g[p][i] ='Q';
            col[i] = dg[i+p] = udg[n+i-p] = 1;
            dfs(p+1);
            dg[i+p] = udg[n+i-p] = col[i] = 0;//还原现场
            g[p][i] ='.';
        }
    }
}
int main(){
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            g[i][j] = '.';
    dfs(0);
}
```

4 数论

4.1 筛质数

4.1.1 线性筛

```
int prime[MAXN],v[MAXN],son[MAXN]; int m=0;//m表示现在筛出m个质数 son是最小质因数
void primes()
{
    for(int i=2;i<N;i++)
    {
        if(v[i]==0)//如果v[i]为0，说明 i 之前没有被筛到过，i 为质数
            {
                prime[++m] = i;
                son[i] = i; //son[i] = j：i的最小质因数为j
            }
        for(int j = 1;j<=m;j++)//遍历小于 i 的所有质数
        {
            int t = prime[j]; //乘起来大于MAXN就跳出循环
            if(t*i > N) break;
            v[i*t] = 1;//标记 i*prime[j] 的最小质因数是prime[j]
            son[i*t] = t;
            if(i%t==0)break;
        }
    }
}
```

4.2 质因数分解

```
//p[]存所有的质因数，c[]存个数
void divide(int n){
    m = 0;
    for(int i=2;i<=sqrt(n);i++){
        if(n % i == 0){
            p[++m] = i,c[m] = 0;
            while(n % i == 0)n /= i,c[m]++;
        }
    }
    if(n > 1)
        p[++m] = n,c[m] = 1;
```

```
}
```

4.3 最大质因数

```
void init()
{
    for(int i = 2; i < N; i++)
    {
        if(prime[i] == 0)
        {
            for(int j = i; j < N; j += i)
            {
                prime[j] = i;
            }
        }
    }
}
```

4.4 正因数集合

```
vector<int> fac[500010];
for(int i=1;i<=n;i++)
    for(int j=1;j<=n/i;j++)
        fac[i*j].push_back(i);
```

4.5 逆元

4.5.1 费马小定理

```
//用快速幂，费马小定理
ll qpow(ll a,ll b) {
    ll ans=1; a=a%mod;
    while(b>0){
        if(b%2==1) ans=(ans*a)%mod;
        b=b/2; a=a*a%mod;
    }
    return ans;
}
ll inv(ll a) {
    return qpow(a, mod-2);
}
```

4.5.2 线性逆推

```
typedef long long ll;
const int N = 1e5 + 5;
ll fac[N], inv[N], fac_inv[N];
void getFac(int n){
    fac[0] = fac_inv[0] = 1;
    for(int i=1;i<=n;i++) fac[i] = fac[i-1] * i % mod;
    inv[1] = 1;
    for(int i=2;i<=n;i++) inv[i] = 1ll * (mod-mod/i)*
        inv[mod%i] % mod;
    for(int i=1;i<=n;i++) fac_inv[i] = fac_inv[i-1] *
        inv[i] % mod;
}
```

4.5.3 扩展欧几里得

```
typedef long long ll;
void extgcd(ll a,ll b,ll& d,ll& x,ll& y){
    if(!b){ d=a; x=1; y=0;}
    else{ extgcd(b,a%b,d,y,x); y-=x*(a/b); }
}
ll inverse(ll a,ll n){
    ll d,x,y;
    extgcd(a,n,d,x,y);
    return d==1?(x+n)%n:-1;
}
```

5 计算几何

5.1 点初始模板

```
#include <cstdio>
#include <cmath>

using namespace std;

const double pi = acos(-1.0);
const double inf = 1e100;
const double eps = 1e-6;
int sgn(double d)
{
    if(fabs(d) < eps) //fabs返回d绝对值
        return 0;
    if(d > 0)
        return 1;
    return -1;
}
int dcmp(double x, double y) //判断浮点数是否相等
{
    if(fabs(x - y) < eps)
        return 0;
    if(x > y)
        return 1;
    return -1;
}
int main() {
    double x = 1.49999;
    int fx = floor(x);//向下取整函数
    int cx = ceil(x);//向上取整函数
    int rx = round(x);//四舍五入函数
    printf("%f %d %d %d\n", x, fx, cx, rx);
    //输出结果 1.499990 1 2 1
    return 0 ;
}
```

5.2 一些基本运算

```
Vector operator + (Vector A, Vector B)//加
{
    return Vector(A.x+B.x, A.y+B.y);
}
```

```
Vector operator - (Point A, Point B)//减
{
    return Vector(A.x-B.x, A.y-B.y);
}
```

```
Vector operator * (Vector A, double p) //乘
{
    return Vector(A.x*p, A.y*p);
}
```

```
Vector operator / (Vector A, double p)//除
{
    return Vector(A.x/p, A.y/p);
}
```

```
bool operator < (const Point& a, const Point& b) //点集升序排列
{
    if(a.x == b.x)
        return a.y < b.y;
    return a.x < b.x;
}
```

```
bool operator == (const Point& a, const Point& b) //等于运算
{
    if(dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0)
        return true;
    return false;
}
```

```
double Dot(Vector A, Vector B) //点乘
{
    return A.x*B.x + A.y*B.y;
}
```

```
double Cross(Vector A, Vector B) //叉乘
{
    return A.x*B.y-A.y*B.x;
}
```

```
double Length(Vector A) //取模
{
    return sqrt(Dot(A, A));
}
```

```
double Angle(Vector A, Vector B) //两向量夹角 弧度制
{
    return acos(Dot(A, B) / Length(A) / Length(B));
}
```

```
double Area2(Point A, Point B, Point C) //俩向量构成平行四边形有向面积
{
    return Cross(B-A, C-A);
}
```

```
Vector Rotate(Vector A, double rad) //向量逆时针旋转后的向量
{//rad为弧度 且为逆时针旋转的角
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}
```

```
Vector Normal(Vector A) //向量逆时针旋转九十度的单位法向量
{//向量A左转90°的单位法向量
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}
```

```
bool ToLeftTest(Point a, Point b, Point c) //判断折线bc是不是向ab的逆时针方向（左边）转向
{
    return Cross(b - a, c - b) > 0;
}
```

5.3 点模板总结

```
struct Point{
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){}
};
typedef Point Vector;
Vector operator + (Vector A, Vector B){
    return Vector(A.x+B.x, A.y+B.y);
}
Vector operator - (Point A, Point B){
    return Vector(A.x-B.x, A.y-B.y);
}
Vector operator * (Vector A, double p){
    return Vector(A.x*p, A.y*p);
}
Vector operator / (Vector A, double p){
    return Vector(A.x/p, A.y/p);
}
bool operator < (const Point& a, const Point& b){
    if(a.x == b.x)
        return a.y < b.y;
    return a.x < b.x;
}
const double eps = 1e-6;
int sgn(double x){
    if(fabs(x) < eps)
        return 0;
    if(x < 0)
        return -1;
    return 1;
}
bool operator == (const Point& a, const Point& b){
    if(sgn(a.x-b.x) == 0 && sgn(a.y-b.y) == 0)
        return true;
    return false;
}
double Dot(Vector A, Vector B){
    return A.x*B.x + A.y*B.y;
}
double Length(Vector A){
    return sqrt(Dot(A, A));
}
double Angle(Vector A, Vector B){
    return acos(Dot(A, B)/Length(A)/Length(B));
}
double Cross(Vector A, Vector B){
    return A.x*B.y-A.y*B.x;
}
double Area2(Point A, Point B, Point C){
```

```
    return Cross(B-A, C-A);
}
Vector Rotate(Vector A, double rad){//rad为弧度 且为逆时针旋转的角
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}
Vector Normal(Vector A){//向量A左转90°的单位法向量
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}
bool ToLeftTest(Point a, Point b, Point c){
    return Cross(b - a, c - b) > 0;
}
```

5.4 线

```
struct Line
{//直线定义
    Point v, p;
    Line(Point v, Point p):v(v), p(p) {}
    Point point(double t){//返回点P = v + (p - v)*t
        return v + (p - v)*t;
    }
};
```

```
//计算两直线交点
//调用前需保证 Cross(v, w) != 0
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w)
{
    Vector u = P-Q;
    double t = Cross(w, u)/Cross(v, w);
    return P+v*t;
}
```

```
//点P到直线AB距离公式
double DistanceToLine(Point P, Point A, Point B)
{
    Vector v1 = B-A, v2 = P-A;
    return fabs(Cross(v1, v2)/Length(v1));
}//不去绝对值，得到的是有向距离
```

```
//点P到线段AB距离公式
double DistanceToSegment(Point P, Point A, Point B){
    if(A == B)
        return Length(P-A);
    Vector v1 = B-A, v2 = P-A, v3 = P-B;
    if(dcmp(Dot(v1, v2)) < 0)
        return Length(v2);
    if(dcmp(Dot(v1, v3)) > 0)
        return Length(v3);
    return DistanceToLine(P, A, B);
}
```

```
//点P在直线AB上的投影点
Point GetLineProjection(Point P, Point A, Point B)
{
    Vector v = B-A;
    return A+v*(Dot(v, P-A)/Dot(v, v));
}
```

```
//点是否在线段上
bool OnSegment(Point p, Point a1, Point a2)
{
    return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) <= 0;
}
```

```
//两线段是否相交
//不允许交点在顶点处
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2){
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1);
    double c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) < 0);
}
//允许在端点处相交
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2){
    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
    double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
    //if判断控制是否允许线段在端点处相交，根据需要添加
    if(!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)
    ){
        bool f1 = OnSegment(b1, a1, a2);
        bool f2 = OnSegment(b2, a1, a2);
        bool f3 = OnSegment(a1, b1, b2);
        bool f4 = OnSegment(a2, b1, b2);
        bool f = (f1|f2|f3|f4);
        return f;
    }
    return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) < 0);
}
```

6 字符串

6.1 函数

```
标准库
sscanf
sscanf(const char *__source, const char *__format, ...) ：从字符串 __source 里读取变量，比如 sscanf(str,"%d",&a) 。

sprintf
sprintf(char *__stream, const char *__format, ...) ：将 __format 字符串里的内容输出到 __stream 中，比如 sprintf(str,"%d",i) 。

strcmp
int strcmp(const char *str1, const char *str2)：按照字典序比较 str1 str2 若 str1 字典序小返回负值，一样返回 0，大返回正值 请注意，不要简单的认为只有 0，1，-1 三种，在不同平台下的返回值都遵循正负，但并非都是 0，1，-1

strcpy
```

```
char *strcpy(char *str, const char *src) : 把 src 中的字符复制到 str 中， str src 均为字符数组头指针，返回值为 str 包含空终止符号 '\0' 。

strncpy
char *strncpy(char *str, const char *src, int cnt) : 复制至多 cnt 个字符到 str 中，若 src 终止而数量未达 cnt 则写入空字符到 str 直至写入总共 cnt 个字符。

strcat
char *strcat(char *str1, const char *str2) : 将 str2 接到 str1 的结尾，用 *str2 替换 str1 末尾的 '\0' 返回 str1 。

strstr
char *strstr(char *str1, const char *str2) : 若 str2 是 str1 的子串，则返回 str2 在 str1 的首次出现的地址；如果 str2 不是 str1 的子串，则返回 NULL 。

strchr
char *strchr(const char *str, int c) : 找到在字符串 str 中第一次出现字符 c 的位置，并返回这个位置的地址。如果未找到该字符则返回 NULL 。

strrchr
char *strrchr(const char *str, char c) : 找到在字符串 str 中最后一次出现字符 c 的位置，并返回这个位置的地址。如果未找到该字符则返回 NULL 。
```

6.2 Manacher

```
const int N = 11000000 + 5;
char s[N], ma[N*2];
int mp[N*2];//mp[i] 为在s中以对应位置为中心的极大子回文串的总长度+1
int Manacher(char *s, int n){
    int len = 0;
    ma[len++] = '$'; ma[len++] = '#';
    for(int i=0;i<n;i++){
        ma[len++] = s[i];
        ma[len++] = '#';
    }
    ma[len] = 0;
    int mx = 0, id = 0;
    int res = 0;
    for(int i=0;i<len;i++){
        mp[i] = mx > i ? min(mp[2*id-i], mx - i) : 1;
        while(ma[i+mp[i]] == ma[i-mp[i]]) mp[i]++;
        if(i + mp[i] > mx) {
            mx = mp[i] + i;
            id = i;
        }
        res = max(res, mp[i] - 1);
    }
    return res;
}
int main(){
    scanf("%s", s);
    printf("%d", Manacher(s, strlen(s)));
    return 0;
}
```

6.3 KMP

```
//以 1 为起点，求nxt数组
nxt[1] = 0;
for(int i=2,j=0; i<=n; i++){
    while(j > 0 && a[i] != a[j+1]) j = nxt[j];
    if(a[i] == a[j+1]) j++;
    nxt[i] = j;
}

//求 f 匹配数组
for(int i=1,j=0;i<=m;i++){
    while(j > 0 && (j == n || b[i] != a[j+1])) j=nxt[j];
    if(b[i] == a[j+1]) j++;
    f[i] = j;
    //if(f[i] == n) 此时为A在B中的某一次出现
}
```

7 背包

7.1 01

```
for(int i = 1; i <= n; ++i)
{
    for(int j = m; j >= 0; --j)
        if(j >= w[i])
            f[j] = max(f[j], f[j-w[i]] + v[i]);
}
```

7.2 完全背包

```
        cin >> n >> m;
        for(int i = 1; i <= n;i++ ) cin >> v[i] >> w[i];
        f[0] = 0;
        for( int i = 1; i <= n; i++){
            for(int j = v[i]; j <= m; j++){
                f[j] = max(f[j] , f[j-v[i]] + w[i]);
            // cout << f[j] << endl;
            }
        }
        for(int i = 0 ; i <= m; i++)
        ans = max(ans , f[i]);
        cout << ans <<endl;
```

7.3 多重背包

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。输出最大价值。

```
    cin>>n>>m;
    while(n--)
    {
    cin>>v>>w>>s;
    while(s--)
    {a[++t]=v;
    b[t]=w;}//死拆，把多重背包拆成01背包
    }
```

```
    for(int i=1;i<=t;i++)
    for(int j=m;j>=a[i];j--)
    dp[j]=max(dp[j-a[i]]+b[i],dp[j]);//直接套01背包的板子
```

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。输出最大价值。

```
    int N,V,v[1001],w[1001],dp[2001],s[1001]
    int a[25000],b[25000]; //2的12次方大于2000，也就是说一个数最多可以拆成12个，故数组容量乘12
    cin>>N>>V;
    memset(dp,0,sizeof(dp));
    for(int i=0;i<N;i++)
     cin>>v[i]>>w[i]>>s[i];
    int total=0;
    for(int i=0;i<N;i++)
    {
        for(int j=1;j<s[i];j<<=1)//二进制拆分
        {
           a[total]=j*w[i];//存价值
           b[total++]=j*v[i];//存容量
           s[i]-=j;
        }
        if(s[i])//当s[i]>0;
        {
            a[total]=s[i]*w[i];
            b[total++]=s[i]*v[i];
        }
    }
    for(int i=0;i<total;i++)//01背包
     for(int j=V;j>=b[i];j--)
      dp[j]=max(dp[j],dp[j-b[i]]+a[i]);
    cout<<dp[V];
    return 0;
}
```

7.4 分组背包

有 N 组物品和一个容量是 V 的背包。

每组物品有若干个，同一组内的物品最多只能选一个。每件物品的体积是 vij，价值是 wij，其中 i 是组号，j 是组内编号。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

```
    for(int i=0;i<n;i++){
        cin>>s[i];
        for(int j=0;j<s[i];j++){
            cin>>v[i][j]>>w[i][j];
        }
    }

    for(int i=0;i<n;i++){
        for(int j=m;j>=0;j--){
            for(int k=0;k<s[i];k++){ //for(int k=s[i];k>=1;k--)也可以
               if(j>=v[i][k]) f[j]=max(f[j],f[j-v[i][k]]+w[i][k]);
            }
        }
    }
```